Efficient Automata-Based Assertion-Checker Synthesis of SEREs for Hardware Emulation

Marc Boulé and Zeljko Zilic

January 24th
ASPDAC 2007
Outline

- Assertion-Based Verification and PSL
- Assertion-Checker Generation
- Automata Synthesis of Sequences (SEREs)
  - 3 Key Algorithms
- Experimental Results
Introduction

- Temporal sequences: crucial for temporal assertion languages such as SVA and PSL
- Need hardware implementation of sequences for resource-efficient assertion checker circuits
- Assertion checkers useful in
  - Hardware emulation
  - Post-fabrication silicon debug
  - On-line monitoring
Assertions

- Assertion Based Verification: Assertion failures used to identify bugs in design
- Assertion: **Formal statement for describing correct behavior of design**
  - Formal (static) or simulation-based (dynamic) verification
  - IEEE 1850 Property Specification Language (PSL), SVA
- Sequential Extended Regular Expressions (SEREs) used to express temporal sequences of events
  - How to implement automata-based SEREs for dynamic verification checkers?
Verification With Assertions

- always \{\sim req; req\} |-> ( \{ [*0:5] ; gnt \} abort \sim rst )
- Does the property hold?
  - Inspect waveform or write code to check the property (often tedious!)

+ Assertion does the monitoring for us
Property Specification Language

Properties

Sequences

Booleans

Base primitives: Boolean expressions

Sequential regular expressions of Booleans

High-level temporal relationships between sequences and Booleans
Property Specification Language

Properties

... Sequences used in conditional and obligation contexts in properties

Sequences

Concatenation, Disjunction, Fusion, Repetition, Goto repetition, Non-consecutive repetition, Intersection

Booleans

HDL Boolean expressions, implication and equivalence, PSL built-in functions
PSL SEREs – Base Operators

- Sequential regular expressions $r$ composed of:
  - $b$ Boolean expressions
  - $\{r\}$ Grouping (like parentheses in RE)
  - $r_1 ; r_2$ Concatenation
  - $r_1 : r_2$ Fusion (overlapped concatenation)
  - $r_1 | r_2$ Disjunction
  - $r_1 \&\& r_2$ Intersection, length-matching
  - $[*0]$ Empty SERE (like $\varepsilon$ in RE)
  - $r[*]$ Kleene closure (like * in RE)

- Example: $\{$busyp*2 ; ack $\} | \{$busyp*5 ; error $\}$
PSL SEREs – Sugaring

- SERE sugaring rules in PSL (non-exhaustive):
  - \( r[^c] = r ; r ; \ldots ; r \)  
    Fixed count repetition, \( c > 0 \)
  - \( r[^l:h] = r[^l] | \ldots | r[^h] \)  
    Bounded repetition
  - \( b[->] = (~b)[^*] ; b \)  
    Goto repetition
  - \( b[->c] = \{b[->]\}[^c] \)
  - \( b[=c] = b[->] ; (~b)[^*] \)  
    Non-consecutive repetition

- Example: \({\{ busy[^*] \} && \{ reset[->] \}}\)

Start matching

Matched
MBAC Checker Generator

- Circuit-level checkers from assertion statements

![Diagram showing MBAC Checker Generator process]

- Device Under Verification
- HDL
- Assertions
- PSL
- MBAC Checker Generator
- Assertion Checkers
- HDL
- A1
- A2
- ... An
- Asr. Fail
- Fail
Checker Generation Process

- Assertion ➔ Finite Automaton ➔ HDL

assert always \( \{a\} \rightarrow \{c[0:1]; d\} \rightarrow \{e\} \)

Diagram:
- State s1 transitions to s2 on input a.
- State s2 transitions to s3 on input c & ~d & ~e.
- State s3 transitions to fail on input ~d.
- State s2 transitions to fail on input ~c & ~d & ~e.

Assertion-Checker Synthesis of SEREs
**SERE Modes in Properties**

- **Conditional Mode**: Identify all occurrences of expression for a given start condition
- **Obligation Mode**: Identify the first failure of expression for a given start condition

- **always** \{\~req;req\} \rightarrow ( \{[*0:3] ; gnt \} )

“\rightarrow” is temporal implication
SEREs vs. Regular Expressions

- SEREs extend traditional REs with:
  - Length-matching intersection
  - Fusion (overlapped concatenation)
  - Based on Boolean expressions (not mutually exclusive symbols as in REs)

- SEREs in properties → failure detection also
  - Obligation mode needed, not only occurrence detection
SERE FA Construction
(Conditional mode)

- Inductive construction [Hopcroft’00]
- Base case: Top level Boolean Expressions $b_i$

Inductive cases:
- Disjunction
- Concatenation
- Kleene closure
- Fusion
- Intersection (length matching)

Custom algorithms → same effect as: NFA Construction [Hopcroft’00] + $\varepsilon$ Removal
SERE FA Construction – Fusion

- Example for \{b_1; b_2[*]\};\{b_3; b_4\}

\[ O(m+n) \]

\[
\begin{align*}
0 & \xrightarrow{b_1} 1 \\
& \xrightarrow{b_2} 1
\end{align*}
\]

\[
\begin{align*}
0 & \xrightarrow{b_3} 1 \\
& \xrightarrow{b_4} 2
\end{align*}
\]

\[
\begin{align*}
0 & \xrightarrow{b_1} 1 \\
& \xrightarrow{b_2 \land b_3} 3 \\
& \xrightarrow{b_1 \land b_3} 4 \\
1 & \xrightarrow{b_2} 1 \\
3 & \xrightarrow{b_3} 4
\end{align*}
\]
**SERE FA Construction – Fusion**

- Example for $\{b_1; b_2[*]\};\{b_3; b_4\}$

\[
\begin{align*}
\text{Example for } \{b_1; b_2[*]\};\{b_3; b_4\} & \quad \text{O(m+n)} \\
\text{m states} & \quad \vdots \\
\text{n states} & \quad = \\
\end{align*}
\]

\[
\begin{align*}
\text{Example for } \{b_1; b_2[*]\};\{b_3; b_4\} & \quad \text{O(m+n)} \\
\text{m states} & \quad \vdots \\
\text{n states} & \quad = \\
\end{align*}
\]
SERE FA Construction – Intersection

Example for \{b_1[^*]; b_2\} \&\& \{b_3, b_4\}

Worst case \(O(mn)\)

State construction stack:

1. 1, C: no edges in C. (1 & C final)
2. 0, C: no edges in C.
3. 0, B: “b_1 \& b_4” 0, C; “b_2 \& b_4” 1, C
4. 1, B: no edges in 1.
5. 0, A: “b_2 \& b_3” 1, B; “b_1 \& b_3” 0, B

ASPDAC 2007  Assertion-Checker Synthesis of SEREs
SERE FA Construction – Intersection

Example for \( \{b_1[^*];b_2\} \& \& \{b_3; b_4\} \)

Worst case \( O(mn) \)

State construction stack:
1,C: no edges in C. (1 & C final)
0,C: no edges in C.
0,B: “b1 \& b4” 0,C; “b2 \& b4” 1,C
1,B: no edges in 1.
0,A: “b2 \& b3” 1,B; “b1 \& b3” 0,B
Obligation Mode SEREs in Properties – FirstFail()

Example for \{b_1[*0:1]; b_2\}

- Strong Determinization
  - Worst case \(O(e^n)\)
- Pseudo Negation:
  - Failure Conditions
Obligation Mode SEREs in Properties – FirstFail()

- Example for \( \{b_1[*0:1]; b_2\} \)

\[
\begin{align*}
0 & \xrightarrow{b_1} 1 & \xrightarrow{b_2} 2 \\
0 & \xrightarrow{b_1 \land \neg b_2} 1 & \xrightarrow{\neg b_2} F
\end{align*}
\]

\[
\begin{align*}
0 & \xrightarrow{b_1} 1 & \xrightarrow{b_1 \land b_2} 1,2 & \xrightarrow{b_2} 2 \\
0 & \xrightarrow{\neg b_1 \land b_2} 2 \\
1,2 & \xrightarrow{b_2} 1
\end{align*}
\]

Strong Determinization

Worst case \( O(e^n) \)

Pseudo Negation:
- Failure Conditions
- Remove old final states
## Experimental Results

<table>
<thead>
<tr>
<th>Properties</th>
<th>MBAC</th>
<th>IBM FoCs 2.03</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Xilinx 8.1.03i for XC2V1500-6)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>never { a;d;{b;a}[*2:4];c;d }</td>
<td>12</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>622</td>
<td>622</td>
</tr>
<tr>
<td>never { a[*];b[1:3] }</td>
<td>{c;d[1:2];e } }</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td>622</td>
<td>454</td>
</tr>
<tr>
<td>never { [[*];a] &amp;&amp; {b[0]} }</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>N.A.</td>
<td>622</td>
</tr>
<tr>
<td>never { a ; {b;c;d } &amp; {e;b;a;d } ; a }</td>
<td>6</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>680</td>
<td>622</td>
</tr>
<tr>
<td>never { {a[<em>]} : {b[</em>]} }</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>680</td>
<td>483</td>
</tr>
<tr>
<td>always {a}</td>
<td>=&gt; { {b;c;d } &amp; {e;d;b } }</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>483</td>
<td>No Output</td>
</tr>
<tr>
<td>always {a}</td>
<td>=&gt; { e;d;{b;e}[2:4];c;d }</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>21</td>
<td></td>
</tr>
<tr>
<td></td>
<td>378</td>
<td>No Output</td>
</tr>
<tr>
<td>always {a}</td>
<td>=&gt; { b ; {c[0:4]} &amp; {d } ; e }</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td></td>
</tr>
<tr>
<td></td>
<td>487</td>
<td>359</td>
</tr>
<tr>
<td>always {a}</td>
<td>=&gt; { b ; {c[0:6]} &amp; {d } ; e }</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td></td>
</tr>
<tr>
<td></td>
<td>428</td>
<td>No Output</td>
</tr>
<tr>
<td>always {a}</td>
<td>=&gt; {{{c;d}[+] &amp; {e[-&gt;2]} }}</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>517</td>
<td>425</td>
</tr>
</tbody>
</table>
Conclusion

- Introduced an efficient automaton-based implementation of SEREs for creating checkers for dynamic verification and silicon debug
  - Boolean-expressions in automata symbols
  - Fusion and intersection algorithms
  - First failure detection algorithm for use in properties
- These techniques for SEREs + property implementation from [HLDVT’06] = efficient assertion checking circuits for PSL